Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (Currently Amended) A method of queuing threads among processors in a multiple processor system having a plurality of multi-processor modules, wherein each of the plurality of multi-processor modules comprises a plurality of processors, wherein each of the plurality of multi-processor modules is associated with one of a plurality of chip run queues, and wherein each of the plurality of processors is associated with one of a plurality of local run queues, the method comprising the computer implemented steps of:

receiving a first thread to be processed, wherein the first thread belongs to a first process;

identifying whether the first thread is a bound thread associated with a first processor of the plurality of processors;

responsive to identifying that the first thread is the bound thread associated with the first processor, assigning the first thread to a first local run queue of the plurality of local run queues, wherein the first local run queue is associated with the first processor;

responsive to not identifying that the first thread is the bound thread, identifying whether the first thread [[as]] is part of an existing process on a first multi-processor module of the plurality of multi-processor modules; [[and]]

responsive to identifying that the first thread is part of the existing process on the first multi-processor module, attempting to identify ~~performing a search for~~ an idle processor that has executed a second thread belonging to the first process, wherein the ~~search is restricted to~~ idle processor is identified from the plurality of processors of the first multi-processor module associated with the existing process; [[and]]

responsive to identifying that the idle processor that has executed the second thread belongs to the first process, assigning the first thread to ~~either one of the plurality of chip~~ a second local run queue of the plurality of local run queues, wherein the second local run queue is associated with the idle processor; and ~~queues, or one of the plurality of local run queues.~~

responsive to not identifying that the idle processor that has executed the second thread belonging to the first process, assigning the first thread to a first chip run queue, wherein the first chip run queue is associated with the first multi-processor module.

2. (Previously Presented) The method of claim 1, further comprising:

assigning the first thread to the chip run queue dedicated to the first multi-processor module.

3. (Original) The method of claim 2, further comprising:

identifying the first multi-processor module as associated with the existing process.

4. (Previously Presented) The method of claim 3, wherein the step of identifying the first multi-processor module further comprises:

maintaining a record of processes having threads executed by a processor of the plurality of processors of the first multi-processor module during a predetermined preceding interval.

5. (Previously Presented) The method of claim 1, further comprising:

identifying one of the plurality of processors of the first multi-processor module as an idle processor; and

assigning the first thread to a local run queue of the plurality of local run queues associated with the idle processor.

6. (Original) The method of claim 1, wherein the step of identifying further comprises:

reading attribute information of the first thread.

7. (Currently Amended) A method of load balancing threads among processors in a multiple processor system having a plurality of multi-processor modules, wherein each of the plurality of multi-processor modules comprises a plurality of processors, wherein each of the plurality of multi-processor modules is associated with one of a plurality of chip run queues, and wherein each of the plurality of processors is associated with one of a plurality of local run queues, the method comprising the computer implemented steps of:

performing, by an idle processor of the plurality of processors of a first multi-processor module of the plurality of multi-processor modules, a first attempt at a thread steal from a first one of the plurality of local run queues of one of the plurality of processors located on the first multi-processor module for reassignment of a thread to a second one of the plurality of local run queue associated with the idle processor;

responsive to performing the first steal attempt, stealing the thread from the first one of the plurality of local run queues if the first one of the plurality of local run queues has a largest number of threads of all the local run queues of the multi-processor module, if the first one of the plurality of local

run queues contains more threads than an intra-module steal threshold of the first multi-processor module, and if the thread is an unbound thread;

responsive to failure of the first attempt, performing a second attempt at [[a]] the thread steal from one of the plurality of chip run queues associated with a second multi-processor module of the plurality of multi-processor modules; [[and]]

responsive to performing the second steal attempt, stealing the thread from the one of the plurality of chip run queues associated with the second multi-processor module of the plurality of multi-processor modules, if the one of the plurality of chip run queues has a largest number of threads of all of the plurality of chip run queues of the multi-processor system, and if the one of the plurality of chip run queues contains more threads than an inter-module steal threshold of the first multi-processor module; and

responsive to stealing the thread from either the first one of the plurality of local run queues or the one of the plurality of chip run queues associated with a second multi-processor module, assigning the [[first]] thread to a chip run queue of the idle processor, or a local run queue of the idle processor ~~either one of the plurality of chip run queues, or one of the plurality of local run queues~~.

8. (Previously Presented) The method of claim 7, further comprising:

evaluating a criterion associated with the second multi-processor module; and

responsive to evaluating the criterion, determining if a thread is to be reassigned from the one of the plurality of chip run queues to the local run queue of the idle processor.

9. (Previously Presented) The method of claim 7, further comprising:

reassigning a thread of the one of the plurality of chip run queues to the local run queue of the idle processor.

10. (Previously Presented) The method of claim 7, further comprising:

responsive to failure of the second attempt, performing a third attempt at a thread steal from one of the plurality of local run queues associated with one of the plurality of the second multi-processor module for reassignment of a thread to the second one of the plurality of local run queues associated with the idle processor.

11. (Previously Presented) A method of load balancing processors in a multiple processor system having a plurality of multi-processor modules, wherein each of the plurality of multi-processor modules comprises a plurality of processors, wherein each of the plurality of multi-processor modules is associated

with one of a plurality of chip run queues, and wherein each of the plurality of processors is associated with one of a plurality of local run queues, the method comprising the computer implemented steps of:

comparing a first thread load of a first chip run queue of the plurality of chip run queues dedicated to a first multi-processor module of the plurality of multi-processor modules with a second thread load of a second chip run queue of a plurality of chip run queues dedicated to a second multi-processor module of the plurality of multi-processor modules; and

reassigning a thread of the first chip run queue to the second chip run queue.

12. (Previously Presented) The method of claim 11, wherein the step of comparing further comprises:

determining a difference between the first thread load of the first chip run queue and the second thread load of the second chip run queue, reassigning the thread responsive to evaluating the difference as greater than a threshold.

13. (Currently Amended) A computer program product in a computer readable recordable-type medium for queuing threads among processors in a multiple processor system having a plurality of multi-processor modules, wherein each of the plurality of multi-processor modules comprises a plurality of processors, wherein each of the plurality of multi-processor modules is associated with one of a plurality of chip run queues, and wherein each of the plurality of processors is associated with one of a plurality of local run queues, the computer program product comprising:

first instructions for receiving a first thread to be processed, wherein the first thread belongs to a first process;; and

~~second instructions for assigning the first thread to a first chip run queue dedicated to a first multi-processor module of the plurality of multi-processor modules.~~

second instructions for identifying whether the first thread is a bound thread associated with a first processor of the plurality of processors;

third instructions, responsive to identifying that the first thread is the bound thread associated with the first processor, for assigning the first thread to a first local run queue of the plurality of local run queues, wherein the a first local run queue is associated with the first processor;

fourth instructions, responsive to not identifying that the first thread is the bound thread, for identifying whether the first thread is part of an existing process on a first multi-processor module of the plurality of multi-processor modules;

fifth instructions, responsive to identifying that the first thread is part of the existing process on the first multi-processor module, for attempting to identify an idle processor that has executed a second

thread belonging to the first process, wherein the the idle processor is identified from the plurality of processors of the first multi-processor module associated with the existing process;

sixth instructions, responsive to identifying that the idle processor that has executed the second thread belongs to the first process, for assigning the first thread to a second local run queue of the plurality of local run queues, wherein the second local run queue is associated with the idle processor; and

seventh instructions, responsive to not identifying that the idle processor that has executed the second thread belongs to the first process, for assigning the first thread to a first chip run queue, wherein the first chip run queue is associated with the first multi-processor module.

14. (Currently Amended) The computer program product of claim 13, further comprising:

[[third]] eighth instructions for identifying a process associated with the first thread, wherein the second instructions identify threads of the process assigned to the first multi-processor module.

15. (Currently Amended) The computer program product of claim 13, further comprising:

[[third]] eighth instructions for comparing a first thread load of the first chip run queue with a second thread load of a second chip run queue dedicated to a second multi-processor module of the plurality of multi-processor modules; and

~~fourth~~ ninth instructions for reassigning the first thread to the second chip run queue.

16. (Currently Amended) The computer program product of claim 13, further comprising:

[[third]] eighth instructions for reassigning the first thread to a first local run queue dedicated to one of the plurality of processors for a second multi-processor module of the plurality of multi-processor modules.

17. (Currently Amended) A multiple processor data processing system for executing multi-threaded processes, comprising:

~~a memory that contains a scheduler as a set of instructions;~~

a first multi-processor module comprising a first plurality of processors and a first chip run queue, wherein each of the first plurality of processors is associated with one of a first plurality of local run queues; [[and]]

a second multi-processor module, comprising a second plurality of processors and a second chip run queue, wherein each of the second plurality of processors is associated with one of a second plurality of local run queues; and ~~wherein the scheduler, responsive to execution of the set of instructions, is~~

~~adapted to receive a thread and assign the thread to a first chip run queue dedicated to the first multi-processor module.~~

a memory that contains a scheduler as a set of instructions, wherein the scheduler, responsive to execution of the set of instructions, is adapted to receive a first thread to be processed, wherein the first thread belongs to a first process; to identify whether the first thread is a bound thread associated with a first processor of the plurality of processors; responsive to identifying that the first thread is the bound thread associated with the first processor, to assign the first thread to a first local run queue of the plurality of local run queues, wherein the first local run queue is associated with the first processor; responsive to not identifying that the first thread is the bound thread, to identify whether the first thread is part of an existing process on a first multi-processor module of the plurality of multi-processor modules; responsive to identifying that the first thread is part of the existing process on the first multi-processor module, to attempt to identify an idle processor that has executed a second thread belonging to a first process, wherein the idle processor is identified from the plurality of processors of the first multi-processor module associated with the existing process; responsive to identifying that the idle processor that has executed the second thread belongs to the first process, to assign the first thread to a second local run queue of the first plurality of local run queues, wherein the second local run queue is associated with the idle processor; and responsive to not identifying that the idle processor that has executed the second thread belongs to a first process, to assign the first thread to the first chip run queue, wherein the first chip run queue is associated with the first multi-processor module.

18. (Original) The data processing system of claim 17, wherein the first multi-processor module comprises a plurality of central processing units disposed on a first chip, and the second multi-processor module comprises a plurality of central processing units disposed on a second chip.

19. (Original) The data processing system of claim 17, wherein the first multi-processor module is a simultaneous multi-threading central processing unit, and the second multi-processor module is a simultaneous multi-threading central processing unit.

20. (Previously Presented) The data processing system of claim 17, wherein the scheduler identifies a second thread of a process associated with the first thread, and the second thread is assigned to the first chip run queue.